Connection Science

Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713411269

Sentence-processing in echo state networks: a qualitative analysis by finite state machine extraction

Stefan L. Frank [a];Henrik Jacobsson [b]
[a] Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, GE, The Netherlands [b] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

First published on: 22 March 2010

To cite this Article Frank, Stefan L. andJacobsson, Henrik(2010) 'Sentence-processing in echo state networks: a qualitative analysis by finite state machine extraction', Connection Science,, First published on: 22 March 2010 (iFirst)
To link to this Article: DOI: 10.1080/09540090903398039
URL: http://dx.doi.org/10.1080/09540090903398039

Connection Science
2010, 1–21, iFirst

Taylor & Francis
Taylor & Francis Group

Sentence-processing in echo state networks: a qualitative analysis by finite state machine extraction

Stefan L. Frank[a]* and Henrik Jacobsson[b]†

Institute for Logic, Language and Computation, University of Amsterdam, P.O. Box 94242, 1090 GE, Amsterdam, The Netherlands; [b]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

(*Received 31 July 2009; final version received 31 August 2009*)

It has been shown that the ability of echo state networks (ESNs) to generalise in a sentence-processing task can be increased by adjusting their input connection weights to the training data. We present a qualitative analysis of the effect of such weight adjustment on an ESN that is trained to perform the next-word prediction task. Our analysis makes use of `CrySSMEx`, an algorithm for extracting finite state machines (FSMs) from the data about the inputs, internal states, and outputs of recurrent neural networks that process symbol sequences. We find that the ESN with adjusted input weights yields a concise and comprehensible FSM. In contrast, the standard ESN, which shows poor generalisation, results in a massive and complex FSM. The extracted FSMs show how the two networks differ behaviourally. Moreover, poor generalisation is shown to correspond to a highly fragmented quantisation of the network's state space. Such findings indicate that `CrySSMEx` can be a useful tool for analysing ESN sentence processing.

Keywords: echo state networks; sentence processing; rule extraction; finite state machines

1. Introduction

Echo state networks (ESNs; Jaeger 2001, 2003) have recently gained popularity as a recurrent neural network (RNN) model for time-series processing. The great advantage of ESNs over the more traditional recurrent networks, such as the simple recurrent network (SRN; Elman 1990), is that the weights of their input and recurrent connections remain fixed at their initial random values. Only the output connection weights are trained, which can be done very efficiently by linear regression, without the need for any iterative gradient-descent method (such as the well-known backpropagation algorithm) for finding proper weights.

Although ESNs have been shown to be useful for several applications, attempts to use them for modelling sentence processing in the field of cognitive science have resulted in mixed outcomes. Whereas one experiment (Tong, Bickett, Christiansen, and Cottrell 2007) showed ESNs to perform

*Corresponding author. Email: S.L.Frank@uva.nl
†Present address: Google, Zurich, Switzerland (since October 2008).

ISSN 0954-0091 print/ISSN 1360-0494 online

DOI: 10.1080/09540090903398039
http://www.informaworld.com

at least as well as SRNs in a next-word prediction task, others (Čerňanský and Tiňo, 2007; Frank 2006a) have found ESNs to generalise relatively poorly.

One way of increasing an ESN's ability to generalise is by adding a hidden layer in-between the recurrent and output layers (Frank 2006a, b). Unfortunately, this also adds another set of connection weights to train, doing away with network's efficient trainability. Alternatively, ESN generalisation can be improved by adjusting the input connection weights to the training input. As shown by Čerňanský, Makula, and Beňušková (2008) and Frank and Čerňanský (2008), this can be done efficiently by one-shot, unsupervised learning. The resulting network can then be trained on the target output just like any ESN, but with greatly increased ability to generalise.

In this paper, we analyse the difference between two types of networks: the standard ESN with random input weights and the alternative network with adjusted input weights (as mentioned above). Following Frank and Čerňanský (2008), we call the latter model ESN+. Both networks are trained on the next-word prediction task in sentence processing. Comparisons between the network's performance on sentences that differ markedly from the training sentences, reported by Frank and Čerňanský, showed that ESN+ generalises much better than does ESN. Here, we extend this *quantitative* comparison with a *qualitative* analysis of networks' behaviour and of the structure of their state spaces. This is done by applying the `CrySSMEx` algorithm (Jacobsson 2006a) for extracting finite state machines (FSMs) from network's input symbols, recurrent-layer states, and output symbols. In this way, the difference between ESN and ESN+ becomes apparent not only from their performance levels, but also from the extracted FSMs, which allows for a more detailed look into the behaviour of the networks. Moreover, the quantisation of the networks' state spaces into FSM states, as computed by `CrySSMEx`, provides insight into the underlying cause of the networks' divergent abilities to generalise.

The next section describes the `CrySSMEx` algorithm. Following this, we present details of our simulations: the language on which the networks were trained and tested, the architecture of the networks, and the difference between ESN and ESN+. Section 4 shows the FSMs that were extracted from the two networks, as well as the corresponding state-space quantisations, at least to the extent that this turned out to be feasible. These results are interpreted and discussed in Section 5.

2. Analysing recurrent neural networks

A fundamental problem for qualitative analyses of RNNs is that even small networks can form complex dynamical systems. Given a single problem instance and a fixed network topology, a wide range of unique behaviours may result from training the network. Understanding each network's idiosyncratic behaviour can take a lot of effort. Alternatively, a statistical analysis (i.e., over populations of RNNs) could be performed. Such an analysis is usually not preferred, however, because the networks' individual characteristics are typically lost.

Countless analysis methods have been developed in parallel with new RNN algorithms and architectures (for a brief list, see Jacobsson 2005). In most cases, trained network instances are analysed on the basis of their hidden activations (e.g., Elman 1990). However, the analysis often requires deep, or even complete knowledge of the domain itself. As a typical and excellent example of this, Bodén and Wiles (2000) used several tools to analyse, quantify, and group qualitatively different RNNs. The networks were trained on context-free and context-sensitive formal languages. Since the RNNs were fed long monotonous sequences, their behaviours were largely governed by the dynamics near the fixed points of the state space. The analysis took advantage of this by using the properties of the eigenvalues of the Jacobian matrix near the fixed point as a basic taxonomy. The analysis was complemented with state-space

plots, which was only feasible because the number of state nodes was limited to two. Although the results were interesting (showing previously unknown regularities in the dynamics of the RNNs), they could only be obtained because the authors defined the domain precisely and (presumably) knew what they were looking for. Moreover, the network's dynamics was visually accessible.

2.1. *Rule extraction from recurrent neural networks*

As discussed above, many researchers employ an analysis method that is tailored for the specific network and domain under investigation. This form of *ad hoc* analysis is perhaps the most common, and may also be necessary to uncover the specifics of the solutions implemented through RNNs; solutions that may very well be counterintuitive and interesting in many ways.

However, one class of analysis methods, *rule extraction*, represents a broader and more generic approach to the problem of neural network analysis and has become a research field in its own right (Andrews, Diederich, and Tickle 1995; Jacobsson 2005). These techniques are typically developed to be portable and as widely applicable as possible (i.e., independent of network type and domain).

The quality of a rule extractor is typically measured by (adapted from Andrews et al. 1995): *rule accuracy*, in terms of how well the rules perform on the test set; *rule fidelity*, that is, how well the rules mimic the behaviour of the RNN; and *rule comprehensibility*, roughly corresponding the readability of the extracted rules and/or the size of the rule set.

Existing techniques for extracting rules (represented as FSM) from RNNs were surveyed by Jacobsson (2005). It turned out that, despite a great deal of diversity, all algorithms have four constituents in common:

(1) Quantisation of the continuous state space of the RNN (e.g., by self-organising maps or k-means), resulting in a discrete set of states.
(2) Generation of internal states and outputs by feeding input patterns to the RNN.
(3) Construction of rules from the observed state transitions, usually resulting in deterministic automata.
(4) Minimisation of the rule set, using some standard algorithm (see Hopcroft and Ullman 1979).

As argued by Jacobsson (2005), none of the existing techniques attempted to merge these four parts in any principled manner. For example, the surveyed techniques implemented the state-space quantisation through standard techniques, completely independent from machine minimisation, even though many clustering algorithms are based on merging of clusters that are equivalent (Mirkin 1996), in a manner much like the merger of computationally equivalent states by the FSM minimisation algorithms. Moreover, rule-extraction techniques could typically only be successfully applied to low-dimensional networks (up to three dimensions). In fact, basic plots of the state space were often more informative than the extracted machines themselves.

2.2. `CrySSMEx`

A basic problem when using standard clustering algorithms for quantising the state space of an RNN, or of any other dynamical system, is that the state space should not primarily be treated as Euclidean space: since the state of an RNN governs its current and future behaviour, small distances in the state space may have a large impact on the functioning of the network, whereas large state-space distances may only affect the network minimally. The algorithm we use here, `CrySSMEx` (the Crystallising Substochastic Sequential Machine Extractor)[1] (Jacobsson 2006a),

does not suffer from this problem because it takes into account the dynamical rather than static geometric properties of the state space.

2.2.1. *Finite state machine extraction*

The `CrySSMEx` algorithm (outlined in Algorithm 1) is based on the combination of the above-mentioned four constituents. As an RNN operates on training or test data, `CrySSMEx` extracts a probabilistic FSM given the network's input, internal states, and outputs. These data (denoted Ω in Algorithm 1) are sampled while the network processes sequences of inputs, as it does during training or testing. Starting with a single-state FSM, `CrySSMEx` iteratively adjusts the FSM by splitting and merging states (i.e., changing the quantisation of the RNN's state space), until the FSM is fully deterministic.

`CrySSMEx`(Ω)
Input: Time-series data, Ω, containing a sequence of input symbols, output symbols, and state vectors generated by the RNN when processing test data
Output: A deterministic machine M mimicking the RNN (if successful)
begin
  Let M be the stochastic machine based on Ω resulting from an unquantized state space (i.e. only one state);
  repeat
    Select state vectors from Ω corresponding to indeterministic states in M;
    Update the state quantiser by splitting the RNN state space according to selected data;
    Create M using new state quantiser;
    if *M has equivalent states* **then**
      Merge equivalent states;
    end
  until *M is deterministic (success) or the number of iterations exceeds a predefined limit* ;
  return M;
end

ALGORITHM 1 A simplified description of the main loop of `CrySSMEx`. M is created from the sequence of observed RNN inputs, outputs, and states contained in Ω by quantisation of the state space. This quantisation is iteratively refined, resulting in a sequence of increasingly accurate FSMs being extracted.

In each iteration, the current FSM's 'weakest' points, which are its most non-deterministic states, are targeted for improvement by actively selecting the corresponding RNN states as the data to be used for refining the quantisation. The algorithm also keeps the extracted model minimal by merging regions in the state space that correspond to states that are computationally equivalent in the FSM.

It was shown (Jacobsson 2006a; Jacobsson, Frank, and Federici 2007) that `CrySSMEx` can efficiently extract probabilistic (or, in favourable circumstances, deterministic) FSMs from RNNs trained on deeply embedded grammars ($\mathbf{a}^n\mathbf{b}^n$), high-dimensional RNNs (10^3 internal nodes), ESNs, and chaotic systems. Moreover, the algorithm was successfully adapted to extract either Moore- or Mealy-type machines (Jacobsson et al. 2007). In a Mealy machine, the output of the network and the extracted machine is defined over transitions between states. In a Moore machine, on the other hand, the output is defined as a function of the current state and input (Hopcroft and Ullman 1979). Depending on the domain and network, either Moore or Mealy models will be

more compact and/or comprehensible. For the current application, we chose to extract Moore machines, since these correspond more closely to the ESNs, whose output is indeed a function of the current state and input.

2.2.2. *State-space quantisation*

To merge and split the state space, `CrySSMEx` creates a number of simple vector quantisers, arranged hierarchically in a graph (Figure 8 in Section 4.3 shows an example). Each split corresponds to a node in the graph, describing how the state space is separated into increasingly smaller regions.

The algorithm that creates this arrangement of quantisers was called the Crystalline Vector Quantiser (CVQ) in Jacobsson (2006a). Roughly speaking, it generates the so-called CVQ graph by creating enough simple vector quantisers to split the state space properly. Consequently, the number of required quantisers is inversely proportional to the quality of their operation for the data set under investigation. The vector quantiser in Jacobsson (2006a) was chosen to be simple and parameter-free, showing that the extracted machines were not impaired by an inappropriate choice of underlying quantisers. However, this quantiser is very sensitive to outliers or skewed distributions, typically making the resulting CVQ graph size larger than would be implied by the number of iterations of `CrySSMEx` alone.

2.2.3. *Scaling properties*

Prior to `CrySSMEx`, rule extraction from RNNs had only been applied to networks with just a handful of internal units. Whether rule extraction is feasible depends of course on the network's size, but even more strongly so on its dynamics: a chaotic network with just one unit requires more FSM states than a very large but non-chaotic network (for a deeper discussion of this issue, see Jacobsson 2006a). `CrySSMEx` can easily handle much larger and more chaotic networks because it generates a sequence of stochastic FSMs that form increasingly precise approximations of the network's behaviour. If obtaining a full (i.e., deterministic) FSM description of the network is not feasible, the algorithm can be stopped after any iteration, yielding an FSM that may at least be sufficiently complete. Moreover, since `CrySSMEx` currently (and quite deliberately) uses very simple vector quantisation for analysing the networks' state space, it is reasonable to assume that the algorithm's scaling properties could be further improved by using optimised algorithms (e.g., support vector machines). However, an investigation of this issue is beyond the scope of the current paper.

It is less clear how well `CrySSMEx` scales up when the number of input and output symbols increases. The problem is essentially one of data scarcity: in every state there will be many possible input symbols for which a prediction needs to be modelled. How to deal with this is an open issue, but as argued elsewhere (Jacobsson 2006b), one could try an active learning approach in which the extraction algorithm interacts with the network directly to gather more data when needed.

3. Method

This section presents the methodological details of our simulations. First, in Section 3.1, we present the language processed by the networks, as well as the specific division into two distinct sets of sentences: those used for training and those used for testing. As explained in Section 3.2, two ESNs were trained to predict the upcoming word at each point in a sentence. That section also

explains how prediction performance is assessed, and how the networks' outputs are discretised for `CrySSMEx` analysis.

3.1. *The language*

3.1.1. *Lexicon*

The semi-natural language used by Frank and Čerňanský (2008) has a lexicon of 26 words, containing 12 plural nouns (e.g., *girls*, *boys*), 10 transitive verbs (e.g., *chase*, *see*), two prepositions (*from* and *with*), one relative clause marker (*that*), and an end-of-sentence marker denoted [*end*], which is also considered a word. The nouns are divided into three groups: female nouns (e.g., *women*), male nouns (e.g., *men*), and animal nouns (e.g., *bats*). As explained below, this distinction is only relevant for distinguishing between training and test sentences. Since the language has only syntax and no semantics, the names of words within each syntactic category are irrelevant and only provided to make the sentences more readable.

3.1.2. *Grammar*

Sentences are generated by the grammar in Table 1. The simplest sentences, such as *mice love cats* [*end*], are just four words long, but longer sentences can be constructed by adding one or more prepositional phrases or relative clauses. Relative clauses come in two types: subject-relative clauses (SRCs, as in *mice that love cats*…) and object-relative clauses (ORCs, as in *mice that cats love*…). Since SRCs can themselves contain a relative clause (as in *mice that love cats that dogs avoid* [*end*]), there is no upper bound to sentence length.

3.1.3. *Training and test sentences*

We adopted not only the language used by Frank and Čerňanský (2008), but also their division of sentences into subsets used for training and testing. The common approach is to set aside a random sample of possible inputs, and train only on the remaining sentences. However, Frank and Čerňanský's specific goal was to investigate whether ESNs can generalise to sentences that contain words occurring at positions they did not occupy during training.[2] Therefore, specific groups of sentences were withheld during training, such that some nouns only occurred in particular grammatical roles: male nouns were banned from subject position and female nouns did not occur in the object position (Table 1 shows which positions are considered to hold subject nouns

Table 1. Probabilistic context-free grammar of the language.

$S \rightarrow NP_{subj}\ V\ NP_{obj}$ [*end*]
$NP_r \rightarrow N_r$ (0.7) | N_r SRC (0.06) | N_r ORC (0.09) | $N_r\ PP_r$ (0.15)
SRC $\rightarrow$ *that* V NP_{obj}
ORC $\rightarrow$ *that* N_{subj} V
$PP_r \rightarrow$ *from* NP_r | *with* NP_r
$N_r \rightarrow N_{fem}$ | N_{male} | N_{anim}
$N_{fem} \rightarrow$ *women* | *girls* | *sisters*
$N_{male} \rightarrow$ *men* | *boys* | *brothers*
$N_{anim} \rightarrow$ *bats* | *giraffes* | *elephants* | *dogs* | *cats* | *mice*
V $\rightarrow$ *chase* | *see* | *swing* | *love* | *avoid* | *follow* | *hate* | *hit* | *eat* | *like*

Variable *r* denotes a noun's grammatical role (subject or object). The probabilities of different productions are equal, except for NP, where they are given in parentheses.

and which hold object nouns). The resulting training set consisted of 5000 sentences, with an average length of 5.93 words.

Frank and Čerňanský (2008) tested networks on eight different types of sentences, but we will restrict our qualitative analysis to just one of these. The test set consisted of all 2700 sentences with structure 'N_{male} V N_{fem} *that* N_{male} V [*end*]'. That is, all test sentences had one ORC, which modified the second noun. Note that animal nouns do not occur in test sentences, and that the roles of male and female nouns are reversed compared with training sentences: male nouns are in the subject position whereas female nouns are objects. This means that all test sentences differ quite strongly from the training sentences. The first word of a test sentence is always a male noun, which never occurred in the sentence-initial position during training. This makes generalisation to test sentences much more challenging than would have been the case if training sentences had been selected by an unrestricted random sample.

3.2. *The echo state networks*

3.2.1. *Architecture*

We investigated the behaviour of two ESNs, with identical architectures. As shown in Figure 1, words are represented locally on the 26-unit input layer that feeds activation to a recurrent layer, which is called the 'dynamical reservoir' (DR) as is common in the ESN literature. The DR also receives its own activation in the previous time step and feeds activation to the output layer. The output layer, like the input layer, has 26 units, each representing one word.

Input. If the word i forms the input to the ESN at time step t, the input activation vector $\mathbf{a}_{\text{in}}(t) = (a_1, \ldots, a_{26})$ has $a_i(t) = 1$ and $a_j(t) = 0$ for all $j \neq i$. The weights of connections from input units to the DR are collected in the 100×26 input weight matrix $\mathbf{W}_{\text{in}}$. In most ESNs, these weights are chosen randomly, but here we use a different approach in which the input weights depend on the training data. In the ESN+ model, most inputs weights are 0, but for DR-units $j = 1, \ldots, 26$, the weight of the connection from input unit i to DR-unit j equals

$$w_{j,i} = 2N \times \frac{N(i,j) + N(j,i)}{N(i)N(j)},$$

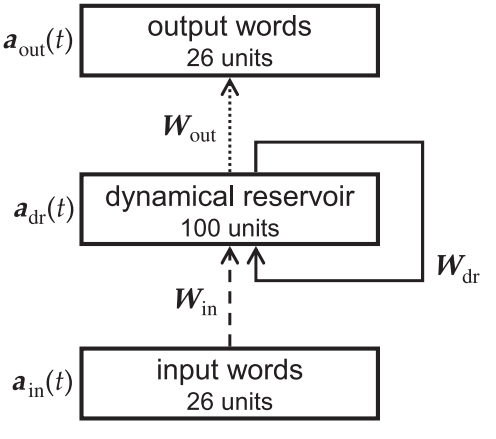

Figure 1. Architecture of ESN(+). Weights of connections between DR units ($\mathbf{W}_{\text{dr}}$; solid arrow) are random and fixed; weights of connection from DR to output units ($\mathbf{W}_{\text{out}}$; dotted arrow) are adapted to training data and task; weights of connections from input to DR units ($\mathbf{W}_{\text{in}}$; dashed arrow) are random and fixed in ESN, but adapted to training data in ESN+.

where N is the number of word tokens in the training data, $N(i)$ is the number of times the word i occurs in the training data, and $N(i, j)$ is the number of times the word j directly follows i. We adopted this particular approach from Frank and Čerňanský (2008), who used it because it was found by Bullinaria and Levy (2007) to yield the word representations that could be successfully applied to several syntactic and semantic tasks. Moreover, it is extremely simple, allowing for fast computation of the input weights.

Choosing $\mathbf{W}_{\text{in}}$ in this manner results in the encoding of syntactic category information in the input weights. If we take column vector i of $\mathbf{W}_{\text{in}}$ (i.e., the weights of connections emanating from input unit i) to represent word i, it turns out that representations of words of the same category cluster together. As Figure 2 shows, there is a clear separation between nouns, verbs, prepositions, *that*, and [*end*]. In spite of the distinctive roles of female, male, and animal nouns in the training sentences, these three groups of nouns are more similar to one another than to other syntactic categories.[3] This increases ESN+ performance on test sentences because words within a syntactic category are equivalent according to the grammar of Table 1 (see Frank and Čerňanský 2008, for a comprehensive discussion).

ESN+ is compared with a standard ESN, that is, one with random input weights. However, a fair comparison between the two networks requires that the ESN's input weights are at least distributed similarly to those of ESN+. For this reason, the ESN's input weights $w_{j,i}$ (for $j = 1, \ldots, 26$) are a random permutation of the corresponding weights of ESN+. For $j > 26$, the ESN's input weights are 0, as in ESN+. Note that this does away with the syntactic information present in $\mathbf{W}_{\text{in}}$, while making sure that the distribution of input weights is the same for ESN and ESN+.

Dynamical reservoir. The two networks, ESN and ESN+, have identical DRs. It has 100 units, connected to each other with weights collected in the 100×100-matrix $\mathbf{W}_{\text{dr}}$. The DR is sparsely connected in that 85% of the values in $\mathbf{W}_{\text{dr}}$ are 0. All other values are taken randomly from a uniform distribution centred at 0, after which they are linearly rescaled such that the spectral radius of $\mathbf{W}_{\text{dr}}$ (i.e., its largest eigenvalue) equals 0.95.

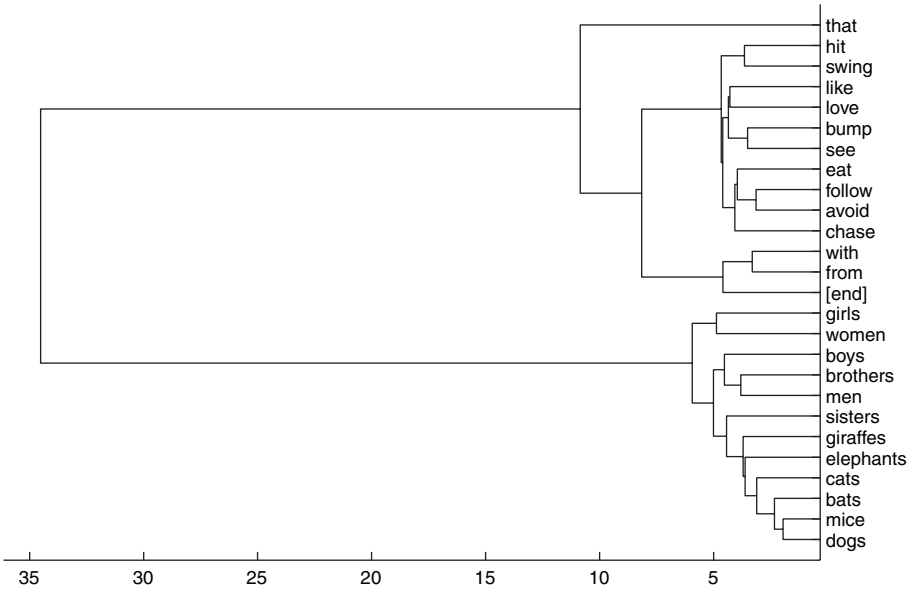

Figure 2. Hierarchical cluster analysis of word representation vectors (i.e., input weights).

The DR's activation vector at time step t, denoted $\mathbf{a}_{\text{dr}}(t) \in [0, 1]^{100}$, is computed at each time step by

$$\mathbf{a}_{\text{dr}}(t) = \mathbf{f}_{\text{dr}}(\mathbf{W}_{\text{dr}}\mathbf{a}_{\text{dr}}(t-1) + \mathbf{W}_{\text{in}}\mathbf{a}_{\text{in}}(t)), \tag{1}$$

where $\mathbf{a}_{\text{dr}}(t-1)$ is the DR state in the previous time step (with $\mathbf{a}_{\text{dr}}(0) = 0.5$ at the beginning of each sentence) and $\mathbf{f}_{\text{dr}}$ is the logistic function.

Output. The weights of connections from DR units to the 26 outputs are collected in the 26×100 matrix $\mathbf{W}_{\text{out}}$. Each output unit i also receives a bias activation b_i. The output at time step t equals

$$\mathbf{a}_{\text{out}}(t) = \mathbf{f}_{\text{out}}\ (\mathbf{W}_{\text{out}}\mathbf{a}_{\text{dr}}(t) + \mathbf{b}), \tag{2}$$

where $\mathbf{b}$ is the vector of bias activations and $\mathbf{f}_{\text{out}}$ is the softmax function:

$$f_{i,\text{out}}(\mathbf{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{3}$$

due to which the outputs are positive and sum to 1, making $\mathbf{a}_{\text{out}}$ a probability distribution. That is, the output value $a_{\text{out},i}(t)$ is the network's estimate of the probability that the word i will be the input at time step $t + 1$.

3.2.2. *Network training*

As explained by Jaeger (2001), useful output connection weights and biases, $\mathbf{W}_{\text{out}}$ and $\mathbf{b}$, are easy to find without any iterative training method. Ideally, a $26 \times (N-1)$ target matrix $\mathbf{U} = (\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(N-1))$ is constructed, where each column vector $\mathbf{u}(t)$ has a value of 1 at the single element corresponding to the input word at $t + 1$, and all other elements are 0. That is, the vector $\mathbf{u}(t)$ forms the correct prediction of the input at $t + 1$. Next, the complete training sequence (excluding the last word) is run through the DR, according to Equation (1). The resulting vectors $\mathbf{a}_{\text{dr}}(t)$ are collected in a $101 \times (N-1)$-matrix $\mathbf{A}$. The connection weight matrix and bias vector are now computed by

$$\begin{aligned} \mathbf{W}_{\text{out}} &= \mathbf{f}_{\text{out}}^{-1}(\mathbf{U})\mathbf{A}^{+}, \\ \mathbf{b} &= \mathbf{f}_{\text{out}}^{-1}(\mathbf{U})\mathbf{1}, \end{aligned} \tag{4}$$

where $\mathbf{A}^{+}$ is the pseudoinverse of $\mathbf{A}$, and $\mathbf{1}$ is an $(N-1)$-element column vector consisting of 1s.

An obvious problem with this method is that the softmax function (Equation (3)) does not have an inverse $\mathbf{f}_{\text{out}}^{-1}$, and even if it did, the values 0 and 1 would be outside the inverse domain. In the past, this problem was avoided by either applying a less efficient training method that does not involve $\mathbf{f}_{\text{out}}^{-1}$ (Frank 2006a, b), or by taking $\mathbf{f}_{\text{out}}$ to be the identity function. In the latter case, the resulting $\mathbf{a}_{\text{out}}$ does not form a probability distribution, so an additional transformation is needed to remove negative values and make sure the activations sum to 1 (Čerňanský and Tiňo 2007, 2008; Čerňanský et al. 2008; Frank and Čerňanský, 2008).

Here, we do take $\mathbf{f}_{\text{out}}$ to be the softmax function and train the network using Equation (4). This is possible because, although $\mathbf{f}_{\text{out}}$ has no inverse in general, a proper $\mathbf{f}_{\text{out}}^{-1}(\mathbf{u})$ does exist for particular target vectors $\mathbf{u}$. In our case, each $\mathbf{u} = (u_1, \ldots, u_n)^{\text{T}}$ has one element u_c arbitrarily close to 1, whereas all other elements have an arbitrarily small but positive value $\epsilon < n^{-1}$ (where n is the number of words in the language, that is, $n = 26$ here). As derived in Appendix 1, the inverse

softmax in that case equals:

$$f^{-1}_{i,\text{out}}(\mathbf{u}) = \begin{cases} \ln(\epsilon^{-1} - n + 1) & \text{if } i = c, \\ 0 & \text{if } i \neq c. \end{cases}$$

We take $\epsilon = 0.01$, making $f^{-1}_{c,\text{out}}(\mathbf{u}) = \ln(75) \approx 4.317$.

3.2.3. *Network analysis*

Prediction performance. Since the grammar of the language is known, the true probability of occurrence of each word at each point in the sentence can be computed. The prediction performance of the networks is rated by comparing their output vectors, which can be interpreted as probability distributions over words, to the true distributions according to the grammar of Table 1. More precisely, we take the Kullback–Leibler divergence from the output distribution to the true distribution as a measure for the network's prediction error.

Extracting FSMs. Extracting an FSM from an RNN requires symbolic input and output, since an FSM's transitions and states come with a finite set of unique labels. The network inputs in our simulations represent words, so denoting them by symbols is straightforward. The networks' outputs, however, are not symbols but probability distributions. These continuous-valued vectors need to be discretised into symbols before `CrySSMEx` can be applied.

The particular choice of output symbols is very important for the analysis. In the extreme case that all network outputs are given the same label, the resulting machine will have only one state and be completely uninformative. In contrast, choosing an output discretisation that is too fine-grained yields 'bloated' and incomprehensible FSMs. Therefore, the granularity of the discretisation should be fine enough for the symbols to be meaningful, but no finer than required for the issue under investigation.

Here, we make use of the fact that all words within each of the five syntactic categories (noun, verb, preposition, *that*, and *[end]*) are equivalent according to the language's grammar (Table 1). This means that syntactic categories are just as meaningful as words, making a descretising of output vectors into words overly fine-grained. We therefore turn these vectors into symbols corresponding to the five syntactic categories. This is done by summing the activations of all output units representing words from the same category. The category with the highest total activation (i.e., the most probable one) is considered to be the output symbol of the network. Summing over words of a category does not give unfair advantage to the categories with many words (such as the nouns) over those with few words (such as the single-word category 'relative clause marker'). This is because, if *some* noun is possible according to the grammar, *any* noun is possible. Therefore, the network needs to spread the total probability mass for 'nouns' over all individual nouns, whereas the total probability of the 'relative clause marker' goes to the single output unit representing the word *that*.

4. Results

4.1. *Network performance*

The networks' prediction error on test sentences is shown in the left panel of Figure 3. As expected, ESN+ does better than the standard ESN. This is true in particular at the sentence-final verb, that is, when having to predict [*end*].

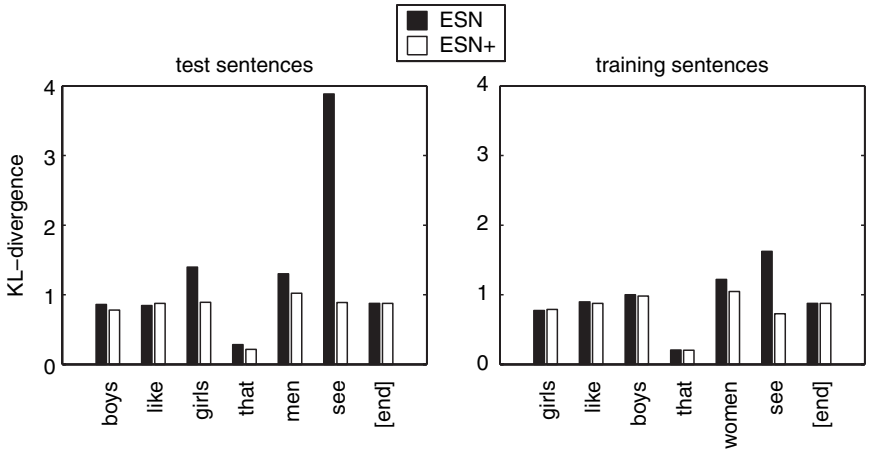

Figure 3. Prediction error of ESN and ESN+ at each word of test sentences (left) and training sentences with the same structure (right). Results are averaged over sentences; the ones shown on the x-axes are just examples.

Eleven of the 5000 training sentences are of the form 'N_{fem} V N_{male} *that* N_{fem} V *[end]*', that is, they have the same structure as the test sentences. The right panel of Figure 3 shows the networks' performance on these training sentences. Again, ESN+ does better than ESN, but the difference is much smaller than it was for test sentences. Interestingly, ESN+ performs nearly as well on test sentences as on training sentences, indicating that it has reached the highest level of generalisation that might be expected. The standard ESN, on the other hand, performs badly at several points of test sentences compared with the same points on training sentences.

4.2. *Extracted FSMs*

The quantitative findings presented above are not particularly surprising as they basically form a replication of Frank and Čerňanský (2008). The contribution of this paper lies in `CrySSMEx`'s qualitative analysis of two trained networks and of the difference between them. Both networks were given all 2700 test sentences as well as all 2700 sentences of the form 'N_{fem} V N_{male} *that* N_{fem} V *[end]*'. These latter sentences could have been in the training set but, as mentioned above, only 11 of them were. During processing of the sentences, the networks' inputs, internal states, and discretised outputs were recorded. Next, `CrySSMEx` was applied to these data, for ESN and ESN+ separately.

4.2.1. *ESN+*

Figure 4 shows the FSM that `CrySSMEx` extracted from the ESN+ data after three iterations. This FSM is fully deterministic, indicating that `CrySSMEx` has converged after this iteration.

The FSM has six states, indicated by the circles numbered 0–5. The machine moves from one state to the next when it receives as input one of the words in the box connecting the two states. At the beginning of a sentence, the FSM must be in State 0 since this is the only state that is entered when receiving the input symbol *[end]*. Moreover, there is no other way to enter this state, showing that it does not occur at any other point in the sentence. When in this state, the network predicts the following input to be a noun, and indeed, all sentences of the language start with a noun, moving the FSM into State 1. Here, it predicts the next word to be a verb, irrespective of whether the input was a male or a female noun. This is remarkable because in the network training data, a verb can only follow a male noun in some SRC constructions, such as *mice that*

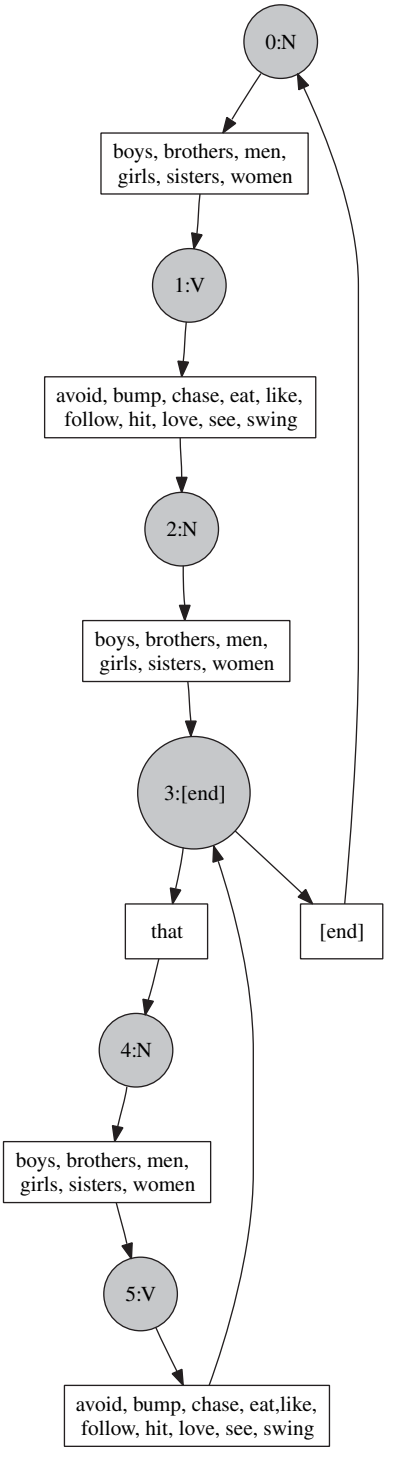

Figure 4. Final finite state machine extracted from ESN+. Circles denote network states. The symbol inside a circle is the output symbol (syntactic category) generated from that state. Words in boxes are the input symbols (words) that move the automaton from one state to the next.

like ***men chase*** *cats [end]*. Only 98 times in the 5000 training sentences did a verb directly follow a male noun, and never at the beginning of a sentence. The correct prediction of a verb in State 1, therefore must result from the previous input being a 'subject' and not just a 'male noun'. That is, ESN+ is more sensitive to the grammatical role of the noun than to its identity. Otherwise, it would not have predicted a verb to come next.

In fact, the FSM of Figure 4 makes no difference whatsoever between male and female nouns: whenever a noun allows for a particular state transition, *all* nouns do. This means that all nouns are equivalent to the FSM, even though male and female nouns are restricted to particular grammatical roles in training sentences. In other words, the network from which the FSM was extracted shows perfect generalisation to test sentences.[4]

After receiving the sentence's second word (a verb), the FSM moves to State 2, where it predicts the next input to be a noun. Indeed a noun occurs next, moving the FSM to State 3. Here, the end-of-sentence marker is predicted, which is incorrect in the sense that the sentence is not yet over. However, it is a grammatically correct prediction: *[end]* can indeed follow 'N V N'.

After receiving *that*, the FSM is in State 4, where the next input is predicted to be a noun, that is, an ORC is expected. It may seem surprising that the FSM never expects an SRC at this point (i.e., it never predicts a verb) even though that would be perfectly grammatical. Presumably, a noun is always expected here because ORCs occur more often than SRCs: according to the grammar of Table 1, ORCs are 50% more frequent than SRCs.

The next two inputs are a noun (moving the system into State 5) and a (correctly predicted) verb. The FSM ends up in State 3, where it correctly predicts *[end]*. It is now back in its starting state. No error (i.e., no grammatically incorrect prediction) was made in processing any of the 5400 sentences.

4.2.2. *Echo state networks*

First iteration. Figure 5 shows the FSM extracted from the ESN data after the first `CrySSMEx` iteration. It has only three states at this point, but `CrySSMEx` did not yet terminate as is apparent from the FSM being stochastic: in many cases, a particular input word licences several transitions from the same state. For example, if the FSM is in State 0 and receives the word *brothers*, it moves to State 1 (where it predicts a verb) in only 40% of the cases. Alternatively, it can remain in State 0 and predict another noun to come next. That prediction is ungrammatical because a noun can never directly be followed by a noun. Yet, in over 29% of the cases, receiving a noun in State 0 results in the ungrammatical prediction of a noun.

Looking at Figure 3, ESN seems to perform only slightly worse than ESN+ after noun inputs. We can now conclude that this is somewhat misleading. The small quantitative advantage of ESN+ over ESN (at noun inputs) in fact corresponds to a very large qualitative improvement: whereas ESN+ makes no errors, ESN often predicts a noun to follow a noun. It is important to keep in mind that this is *not* an error of the extracted FSM. It correctly describes the behaviour of the ESN, which erroneously predicts a noun to follow a noun. Although the FSM is correct in this sense, it is not complete: for instance, it must be in State 0 at the beginning of a sentence (i.e., after processing *[end]*) but that same state can also be entered after processing a noun or verb.

Another error that can be observed in the FSM of Figure 5 is the occasional prediction of a verb (i.e., being in State 1) after processing a verb. Although the language does allow for a verb to be directly followed by another (e.g, *mice that men* ***like chase*** *cats [end]*), this is not possible in any of the 'N V N *that* N V *[end]*'-sentences processed by the FSM. Possibly, this error accounts for ESN's low prediction performance (Figure 3) at the sentence-final verb, that is, when *[end]* is the only correct prediction. Further insight into this error can be obtained from the FSM extracted at the second `CrySSMEx` iteration.

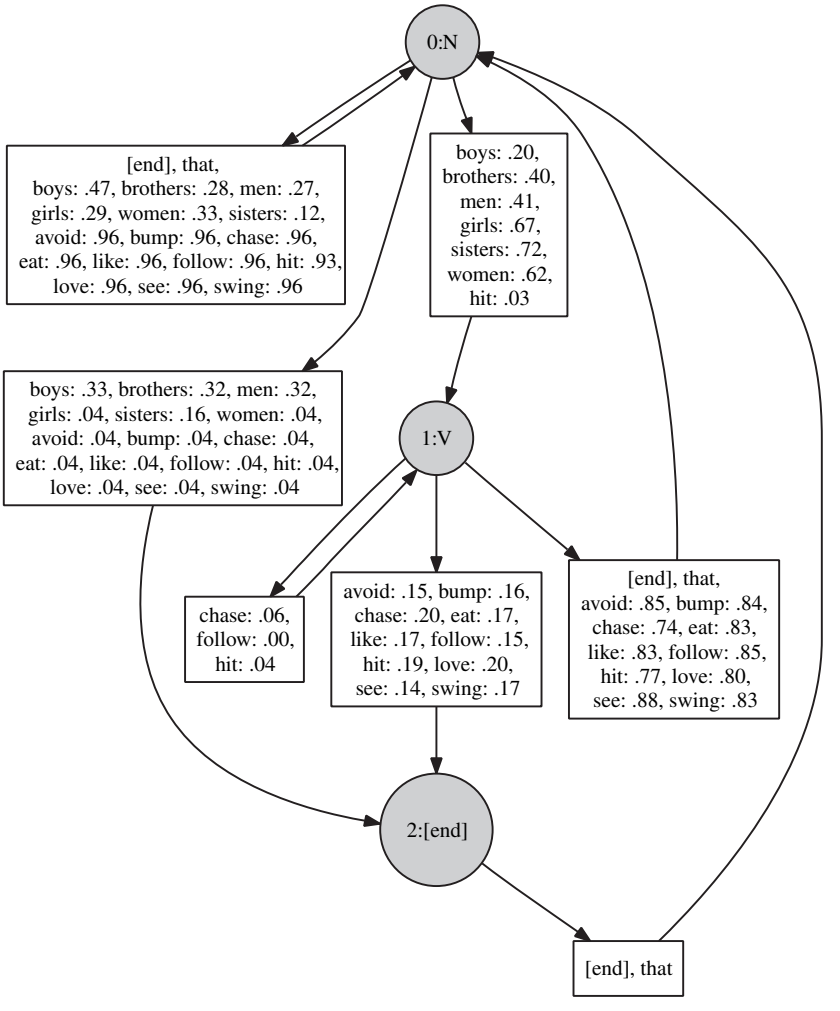

Figure 5. First (stochastic) finite state machine extracted from ESN. If a particular input word licences more than one transition from a state, the probability of a transition is given after the word.

Second iteration. In its second iteration, `CrySSMEx` splits the three states into seven, resulting in the FSM shown in Figure 6. The fact that it is not deterministic shows that `CrySSMEx` has not yet converged, so more iterations are needed for further refinement.

Although this FSM has only one more state than the one extracted from the ESN+ data (Figure 4), it is much more complex, making it difficult to interpret in full. For this reason, we will not discuss it in depth, but only point out how it sheds light on ESN's low prediction performance at the sentence-final verb. Also, the FSM shows why there is a large difference between the performance on training and test sentences at this point.

To begin with, note that State 0 must be the sentence-initial state, as it is the only state entered after the input *[end]*. If the input is a potential training sentence (i.e., 'N_{fem} V N_{male} *that* N_{fem} V *[end]*'), the path through the machine's states is easy to follow. The first word, a female noun, must move the FSM into State 1, where a verb is correctly predicted. The incoming verb will nearly always result in State 2, predicting a noun. The following input is a male noun, moving the FSM into State 6. Here, the prediction *[end]* is grammatical, although the actual next input is *that*, resulting in State 3 in as much as 95% of the cases. The incoming female noun will most

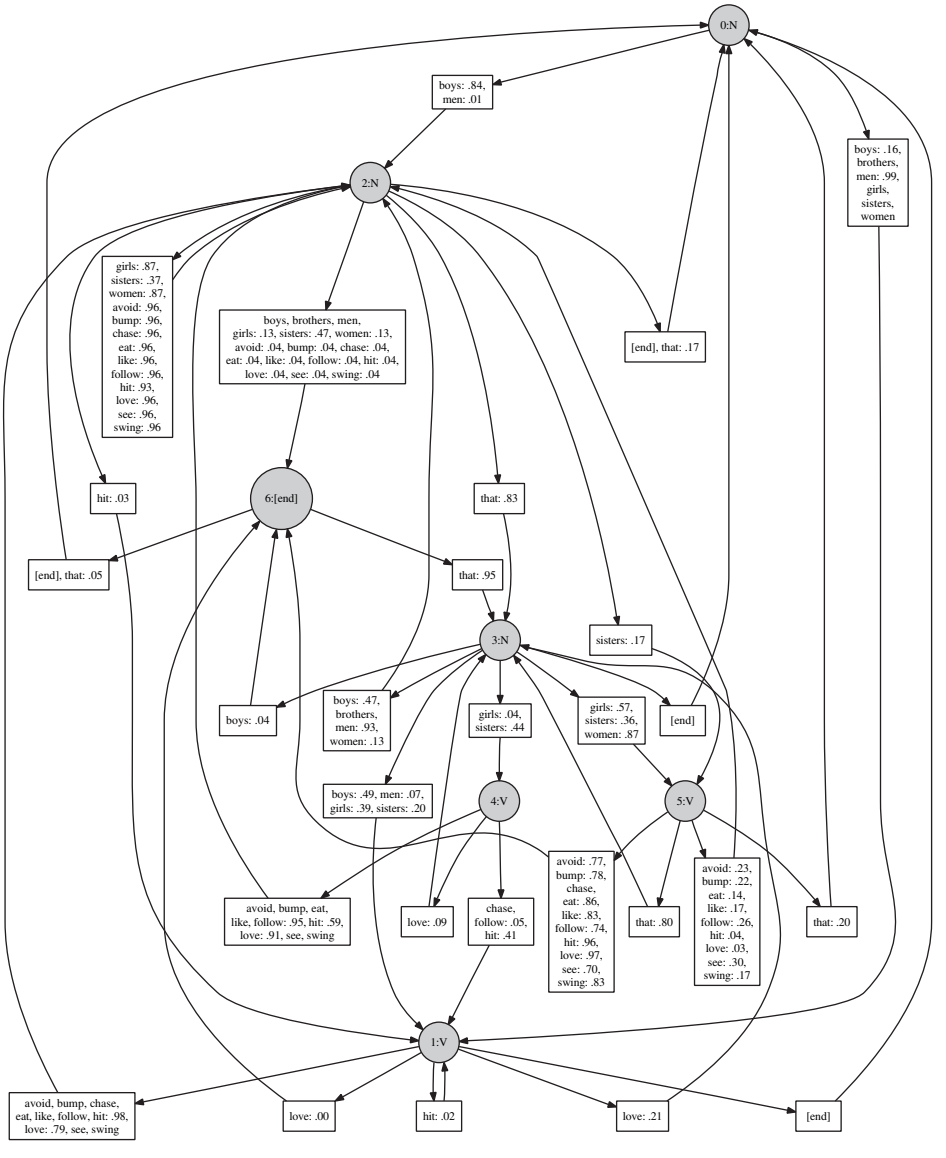

Figure 6. Second finite-state machine extracted from ESN.

often put the FSM into State 5, although States 2 and 4 are also possible. At this point, the only grammatically correct prediction is *[end]*, that is, the FSM should move into State 6. Indeed, a large majority of verb inputs at State 5 yield State 6, but some errors (i.e, a noun prediction at State 2) are also possible. If the FSM was not in State 5 but in State 4, it cannot end up in the correct State 6. Likewise, if the machine was in State 2 (rather than 4 or 5) a verb input is very unlikely to move it to State 6.

In short, there is not much opportunity for errors in training sentences, except at the sentence-final verb. This finding corresponds to ESN's prediction performance on training sentences, as displayed in Figure 3. But how about test sentences? Be reminded that after processing the sentence-final verb, the FSM should be in State 6. This state can be entered from States 1–3 and 5, but only from State 5 it is likely that verb input results in State 6. From States 1–3, there is only a

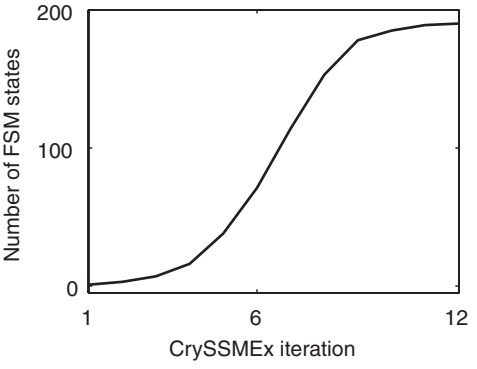

Figure 7. Number of states in the FSMs extracted from ESN, after each CrySSMEx iteration.

very small probability that a verb moves the FSM into State 6. Therefore, we can safely conclude that the machine should be in State 5 immediately *before* the sentence-final verb arrives. However, note that State 5 can only be entered when the input is a female noun whereas the actual input at this point in test sentences is always a *male* noun. Hence, when the input is a test sentence, the FSM is not in State 5 when it needs to be. Consequently, it will hardly ever predict *[end]* when it needs to. This explains the large difference in ESN prediction error between training and test sentences at the sentence-final verb.

Further iterations. As CrySSMEx continues to extract increasingly deterministic FSMs from the ESN, the number of states rises sharply, as can be seen in Figure 7. After 12 iterations, CrySSMEx has converged at an FSM with as many as 190 states.

4.3. *State-space quantisation*

So far, we have only looked into the extracted machines themselves. Machines that, to a large extent, reflect the network behaviour and grammatical correctness rather than their internal dynamics. In addition to these FSMs, CrySSMEx generates hierarchical descriptions of the networks' state space quantisations. Since these CVQ graphs form a rough description of the layout of the state space, they potentially hold important qualitative information about the networks' dynamics.

The CVQ graph describing the state space of ESN+, displayed in Figure 8 shows that ESN+ is trivially mapped onto an FSM: it takes at most five quantisations to determine the FSM state for any state vector in the network. For the state space of ESN, the situation is remarkably different. Figure 6 shows just a small part of the CVQ graph corresponding to the first FSM extracted from the ESN data (i.e., the one in Figure 5). This FSM has fewer states than the final machine extracted from ESN+, yet the CVQ graph is immensely more complex. In short, this tells us that the state space of ESN is much more fragmented than that of ESN+. Consequently, CrySSMEx needs to work a lot harder to render an FSM description for ESN than for ESN+.

5. Discussion

5.1. *Comparing network behaviours*

We have presented the first ever application of CrySSMEx for a qualitative comparison of the behaviours of two networks. The algorithm proved to provide more insight than was obtained from a merely quantitative investigation of network performance. The difference in performance

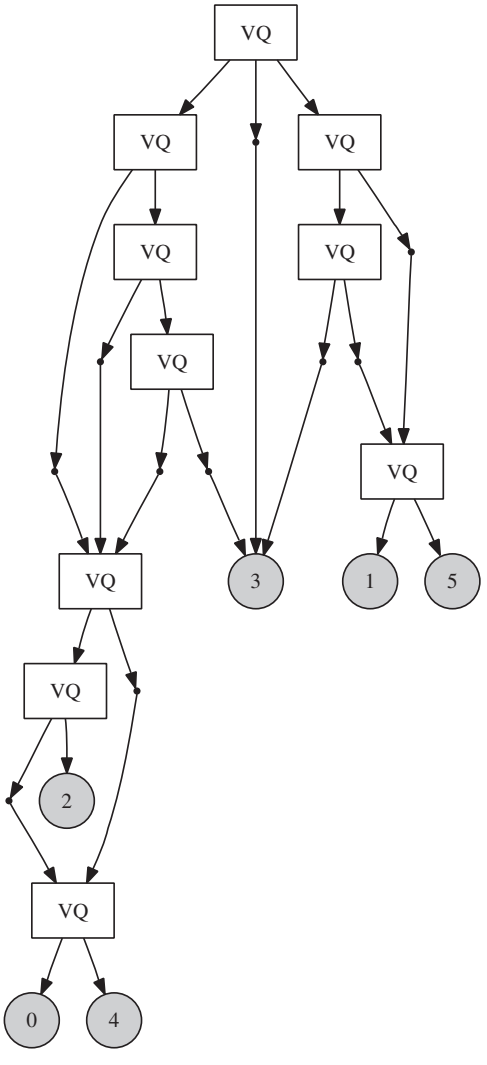

Figure 8. Arrangement of Vector Quantisers (VQ) for splitting the ESN+ state space (corresponding to the FSM of Figure 4). The FSM state to which a given state vector belongs is decided by starting at the root VQ and following the graph according to the winning model vector of each VQ. Arrows with a dot denote a merger of states.

between ESN and ESN+ turned out to be the consequence of a much more significant, qualitative difference in their behaviours. Comparing the two extracted FSMs in Figures 4 and 6, it becomes clear that the two networks implement very different machines. In fact, it is hard to imagine that the RNNs that gave rise to these FSMs are nearly identical and process the same input sentences. The only difference between ESN+ and ESN is that the first has input connection weights that were adapted to the training data, whereas the second uses a random permutation of these. Apparently, this small difference has immense consequences for the networks' dynamics and behaviour: ESN+ implements a straightforward FSM with six states, whereas the FSM extracted from ESN has 190. Even if we restrict ourselves to the non-deterministic, seven-state FSM after the second iteration of `CrySSMEx`, the behaviour of ESN already turns out to be much more complex than that of ESN+.

Looking at Figures 6 and 7, it becomes clear that a system that should be relatively simple (processing just one type of seven-word sentence) can implement an oversized FSM. The primary

reason why the ESN data causes the extracted FSMs to grow so large is that CrySSMEx will blindly keep refining the extracted rules, also with respect to undesired behaviour by the network. In this case, the desired grammar, as instantiated by ESN+, is computationally much simpler than the erratic one instantiated by ESN.

Since CrySSMEx is ignorant about which behaviour is desired and which is not, it cannot be blamed for creating huge FSMs. It merely shows us the networks' behaviour. The more complex that behavior, the more complex the extracted FSM. Thanks to the iterative approach of the algorithm, however, the FSMs extracted in the first few iterations can be quite informative. Although they are incomplete, they may provide a comprehensible 'summary' of the complete, but incomprehensible, deterministic FSM that is implemented by the network.

5.2. *Comparing network state spaces*

In an early comparison between SRNs and FSMs (Servan-Schreiber, Cleeremans, and McClelland 1991), the term *graded state machine* was tossed to describe the type of system embodied by an SRN. The difference with an FSM is that a graded state machine has (possibly infinite) non-discrete states. Each point in an SRN's state space can be considered a state of the machine, and states that are near to each other in the state space tend to (but do not necessarily) result from similar inputs and have similar effects on the network.

ESN+ makes uses of this graded nature of the state space by using inputs that are not truly symbolic. As pointed out by Frank and Čerňanský (2008), the vector of weights emanating from a particular input unit can be viewed as the representation of the word corresponding to that input, and adapting these weights results in representations that are *analogical* rather than symbolic: similarities among word representations are analogical to similarities among the grammatical properties of the represented words. More specifically, words from the same syntactic category are represented by vectors that are more similar to each other than vectors representing words from different categories, as is also apparent from Figure 2.

Each input word drives the activation pattern in ESN's dynamical reservoir towards an attractor point that is unique for that input (Tinŏ, Čerňanský and Beňušková 2004). Because of the analogical nature of word representations in ESN+, the attractor points associated with words from the same syntactic category will be closer together than those of words from different categories. As a result, FSM states that are functionally equivalent correspond to ESN+ state-space points that are near to one another. Such clustering of equivalent states facilitates state-space quantisation. Presumably, this is why CrySSMEx converges after just three iterations when processing the ESN+ data. Moreover, the state-space clustering improves generalisation. This is because processing a test sentence leads to a state-space trajectory that visits the same clusters that were encountered during ESN+ training. In other words, new sentences result in not-so-new internal network states.

For ESNs, however, the situation is radically different. Since its input weights are fixed at random values (i.e., it uses symbolic rather than analogical word representations) the distribution of attractors in its state space is basically random. Even two states that are functionally equivalent can correspond to distant points in the state space. As a result, many splits and merges are required for a meaningful quantisation, delaying CrySSMEx convergence and resulting in the highly complex CVQ graph, part of which is displayed in Figure 9. Also, the generalisation is impaired because the state-space trajectory resulting from a new sentence is largely unrelated to what was encountered during ESN training. To summarise, we have found the impoverished generalisation of ESN to be related to the relative complexity of its internal dynamics, as apparent from the size of its CVQ graph.

As mentioned in section 2.2.2, the depth and size of CVQ graphs are partly governed by the quality of the underlying quantisation algorithm: the worse the quantiser, the larger the CVQ

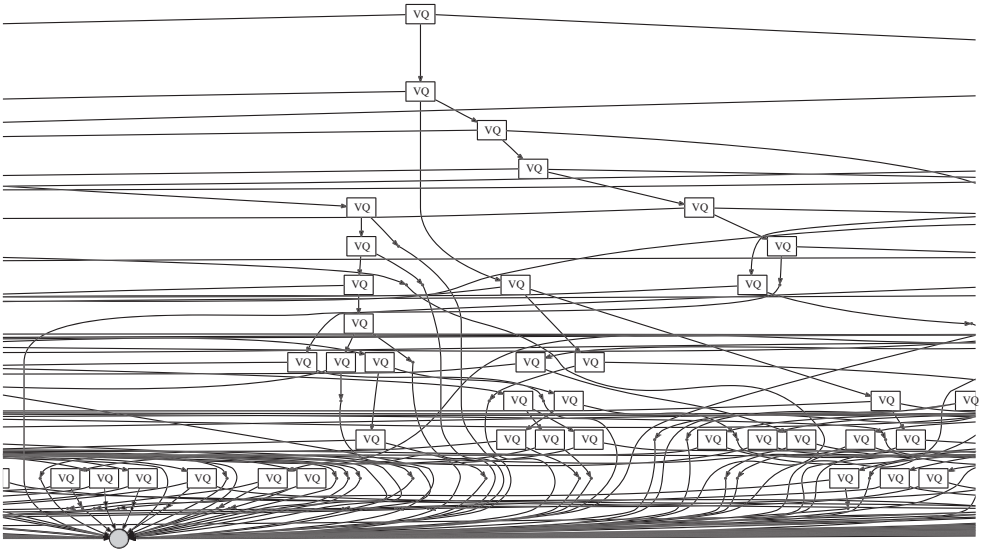

Figure 9. Small fragment of the arrangement of Vector Quantisers (VQ) for splitting the ESN state space (corresponding to the FSM of Figure 5).

graph. Hence, the size of the CVQ graph for ESN (Figure 9) is partly due to the simplicity of the quantiser that was used, rather than from the actual dynamics of the system. However, the immense difference between the CVQ graphs for ESN and ESN+ cannot all be blamed on the quantiser's suboptimality. Therefore, even without a more sophisticated quantisation algorithm, we can provide a reasonable suggestion about what goes on inside the network, and how the dynamics of the two networks underlies their quantitative and qualitative differences.

6. Conclusion

`CrySSMEx` allows finite state descriptions to be generated from high-dimensional and complex ESNs, opening up a new window into the internal workings of specific ESN instances. In this paper, we have only scraped the surface of the interesting dynamics of ESNs. Only a small step was made towards understanding exactly how and why ESN+ manages to utilise its state space in a manner that governs a deeper correspondence to the intended grammar. It is our hope and conjecture that `CrySSMEx` may provide much deeper insights into these systems than presented in this paper, insights that may lead to further improvements beyond those of the ESN+ model.

Acknowledgements

This research was supported by grant 277-70-006 from the Netherlands Organisation for Scientific Research (NWO) and by EU grant FP6-004250-IP.

Notes

1. `CrySSMEx` can be downloaded from `http://cryssmex.sourceforge.net/`
2. This is because it has been argued that people only need to experience a word in one grammatical position to generalise it to novel positions. Consequently, neural networks should also have this ability in order to be regarded as cognitive models of human language processing (Hadley 1994).

3. Note that all nouns within each of the three subcategories (as well as all verbs) are interchangeable in the training sentences. As a result, their representations will become more and more similar as the number of training sentence is increased.
4. That is, when outputs are discretised by syntactical category. As Figure 3 shows, the output vectors of ESN+ are not identical to the true probability distributions.

References

Andrews, R., Diederich, J. and Tickle, A.B. (1995), 'Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks', *Knowledge Based Systems*, 8, 373–389.

Bodén, M. and Wiles, J. (2000), 'Context-free and Context-Sensitive Dynamics in Recurrent Neural Networks', *Connection Science*, 12, 196–210.

Bullinaria, J.A. and Levy, J.P. (2007), 'Extracting Semantic Representations from Word Co-Occurrence Statistics: A Computational Study', *Behavior Research Methods*, 39, 510–526.

Čerňanský, M. and Tiňo, P. (2007), 'Comparison of echo state networks with simple recurrent networks and variable-length markov models on symbolic sequences', in *Artificial Neural Networks – ICANN 2007, Part I (vol. 4668)*, eds. J.M. de Sá, L.A. Alexandre, W. Duch and D.P. Mandic, *Lecture Notes in Computer Science*, Berlin: Springer, pp. 618–627.

Čerňanský, M. and Tiňo, P. (2008), 'Processing Symbolic Sequences Using Echo-State Networks', in *From associations to rules: Proceedings of the 10th Neural Computation and Psychology Workshop*, eds. R.M. French, and E. Thomas, Singapore: World Scientific, pp. 153–159.

Čerňanský, M., Makula, M. and Beňušková Ľ. (2008), 'Improving the state space organization of untrained recurrent networks', in *Proceedings of the 15th International Conference on Neural Information Processing*, Auckland, New Zealand.

Elman, J.L. (1990), 'Finding Structure in Time', *Cognitive Science*, 14, 179–211.

Frank, S.L. (2006a), 'Learn More by Training Less: Systematicity in Sentence Processing by Recurrent Networks', *Connection Science*, 18, 287–302.

Frank, S.L. (2006b), 'Strong Systematicity in Sentence Processing by an Echo State Network', in *Artificial Neural Networks – ICANN 2006, Part I* (vol. 4131), eds. S. Kollias, A. Stafylopatis, W. Duch, and E. Oja, Lecture Notes in Computer Science, Berlin: Springer, pp. 505–514..

Frank, S.L. and Čerňanský, M. (2008), 'Generalization and systematicity in echo state networks', in *Proceedings of the 30th Annual Conference of the Cognitive Science Society*, eds. B.C. Love, K. McRae, and V.M. Sloutsky, Austin, TX: Cognitive Science Society, pp. 733–738.

Hadley, R.F. (1994), 'Systematicity in Connectionist Language Learning' *Mind and Language*, 9, 247–272.

Hopcroft, J. and Ullman, J.D. (1979), *Introduction to Automata Theory, Languages, and Compilation*, Reading, MA: Addison-Wesley Publishing Company.

Jacobsson, H. (2005), 'Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review', *Neural Computation*, 17, 1223–1263.

Jacobsson, H. (2006a), 'The crystallizing substochastic sequential machine extractor: Cryssmex', *Neural Computation*, 18, 2211–2255.

Jacobsson, H. (2006b), 'Rule Extraction from Recurrent Neural Networks', uppublished doctoral dissertation, Department of Computer Science, University of Sheffield, Sheffield, UK.

Jacobsson, H., Frank, S.L. and Federici, D. (2007), 'Automated abstraction of dynamic neural systems for natural language processing', in *Proceedings of the International Joint Conference on Neural Networks*, Orlando, FL, pp. 1446–1451.

Jaeger, H. (2001), 'The "Echo State" Approach to Analysing and Training Recurrent Neural Networks', GMD Report No. 148, GMD – German National Research Institute for Computer Science, http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf.

Jaeger, H. (2003), 'Adaptive Nonlinear System Identification with Echo State Networks', in *Advances in Neural Information Processing Systems* (Vol. 15), eds. S. Becker, S. Thrun and K. Obermayer, Cambridge, MA: MIT Press, pp. 593–600.

Mirkin, B. (1996), *Mathematical Classification and Clustering* (Vol. 11), Dordrecht, The Netherlands: Kluwer Academic Publishers.

Servan-Schreiber, D., Cleeremans, and A. McClelland, J.L. (1991), 'Graded State Machines: The Representation of Temporal Contingencies in Simple Recurrent Networks', *Machine Learning*, 7, 161–193.

Tiňo, P., Čerňanský, M. and Beňušková, Ľ. (2004) 'Markovian architectural bias of recurrent neural networks', *IEEE Transactions on Neural Networks*, 15, 6–15.

Tong, M.H., Bickett, A.D., Christiansen, E.M. and Cottrell, G.W. (2007), 'Learning Grammatical Structure with Echo State Networks', *Neural Networks*, 20, 424–432.

Appendix 1. Inverse of softmax

The softmax function (Equation (3)) does not have an inverse in general. However, it is possible to define a proper inverse for the particular target vectors that arise in our ESN training procedure.

We have a network with n output units, one of which (called unit c) represents the correct output for each input vector. In the corresponding target output vector $\mathbf{u} = (u_1, \ldots, u_n)$, element c should have large value (i.e., close to 1) whereas all other elements should have small values (i.e., close to 0). That is

$$u_j = \begin{cases} 1 - \epsilon_c & \text{if } j = c \\ \epsilon_i & \text{if } j \neq c. \end{cases}$$

Ideally, $\epsilon_c = \epsilon_i = 0$, making $u_c = 1$ and $u_i = 0$. However, the softmax inverse will be applied to $\mathbf{u}$ and its domain does not contain 0 and 1. Therefore, we take $\epsilon_c, \epsilon_i > 0$. Also, it is desired that $u_c > u_i$, so $1 - \epsilon_c > \epsilon_i$. From here on, we denote by i all output units that are not c, so always $i \neq c$. Note that we assume that the target values ϵ_i are equal for all i.

We are looking for the inverse of the softmax function, that is, we want to find x_i and x_c such that

$$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} = \epsilon_i \quad \text{and} \quad \frac{e^{x_c}}{\sum_{j=1}^{n} e^{x_j}} = 1 - \epsilon_c. \tag{A1}$$

First, note that $\sum_{j=1}^{n} e^{x_j} = e^{x_c} + (n-1)e^{x_i}$. Therefore, Equation (A1) becomes

$$\begin{aligned} & e^{x_i} = \epsilon_i e^{x_c} + \epsilon_i (n-1) e^{x_i} \quad \text{and} \quad e^{x_c} = (1-\epsilon_c) e^{x_c} + (1-\epsilon_c)(n-1) e^{x_i} \iff \\ & e^{x_c} = e^{x_i} \left(\frac{1 - n\epsilon_i + \epsilon_i}{\epsilon_i} \right) \quad \text{and} \quad e^{x_c} = e^{x_i} \left(\frac{(1-\epsilon_c)(n-1)}{\epsilon_c} \right) \iff \\ & x_c = \ln \left(\frac{1 - n\epsilon_i + \epsilon_i}{\epsilon_i} \right) + x_i \quad \text{and} \quad x_c = \ln \left(\frac{(1-\epsilon_c)(n-1)}{\epsilon_c} \right) + x_i. \end{aligned} \tag{A2}$$

Since $u_c > u_i$, we must have $x_c > x_i$, so

$$\frac{1 - n\epsilon_i + \epsilon_i}{\epsilon_i} > 1 \iff \epsilon_i < n^{-1}.$$

From Equation A2, it is clear that

$$\frac{1 - n\epsilon_i + \epsilon_i}{\epsilon_i} = \frac{(1-\epsilon_c)(n-1)}{\epsilon_c},$$

which yields $\epsilon_c = (n-1)\epsilon_i$. This means that ϵ_i and ϵ_c cannot be set independently: choosing some minimum target value $\epsilon_i < n^{-1}$ fixes ϵ_c as well. The total target output is

$$\begin{aligned} \sum_j u_j &= (1 - \epsilon_c) + (n-1)\epsilon_i \\ &= 1 - (n-1)\epsilon_i + (n-1)\epsilon_i = 1. \end{aligned}$$

Therefore, the target vector forms a probability distribution.

By choosing a low enough value for ϵ_i, the difference between x_c and x_i is fixed as in Equation A2: $x_c = \ln(\epsilon_i^{-1} - n + 1) + x_i$. As it turns out, it is only this difference that matters in practice: adding a constant value y to x_c and x_i does not change the networks' output. Let $\mathbf{W}_{y,\text{out}}$ denote the output connection weights resulting from this addition of y (for convenience, we ignore the bias vector). As in Equation (4) of Section 3.2.2, $\mathbf{A}$ is the matrix of DR states resulting from the training inputs, and $\mathbf{U}$ is the matrix of corresponding target outputs. The resulting connection weights are

$$\begin{aligned} \mathbf{W}_{y,\text{out}} &= \left(\mathbf{f}_{\text{out}}^{-1}(\mathbf{U}) + y \right) \mathbf{A}^+ \\ &= \mathbf{f}_{\text{out}}^{-1}(\mathbf{U}) \mathbf{A}^+ + y\mathbf{A}^+ \\ &= \mathbf{W}_{\text{out}} + y\mathbf{A}^+. \end{aligned}$$

After training, the network receives an input resulting in the DR state $\mathbf{a}_{\text{dr}}$. Its output becomes (see Equation (2) in Section 3.2.1):

$$\mathbf{f}_{\text{out}}(\mathbf{W}_{y,\text{out}} \mathbf{a}_{\text{dr}}) = \mathbf{f}_{\text{out}}(\mathbf{W}_{\text{out}} \mathbf{a}_{\text{dr}} + y|\mathbf{A}^+ \mathbf{a}_{\text{dr}}|),$$

which equals $\mathbf{f}_{\text{out}}(\mathbf{W}_{\text{out}} \mathbf{a}_{\text{dr}})$ because the softmax function is translation invariant (i.e., $f_{j,\text{out}}(\mathbf{x} + a) = f_{j,\text{out}}(\mathbf{x})$). Since it does not matter whether the connection weights are $\mathbf{W}_{\text{out}}$ or $\mathbf{W}_{y,\text{out}}$, adding y to x_c and x_i has no effect. We can therefore simply take $x_i = 0$.

To summarise, all that is needed for training an ESN with the softmax output activation is to choose an $\epsilon_i < n^{-1}$. The softmax inverse of the target outputs then equals:

$$f_{j,\text{out}}^{-1}(\mathbf{u}) = \begin{cases} \ln(\epsilon_i^{-1} - n + 1) & \text{if } j = c \\ 0 & \text{if } j \neq c. \end{cases}$$